Intelligent Systems (AI-2)

Computer Science cpsc422, Lecture 10

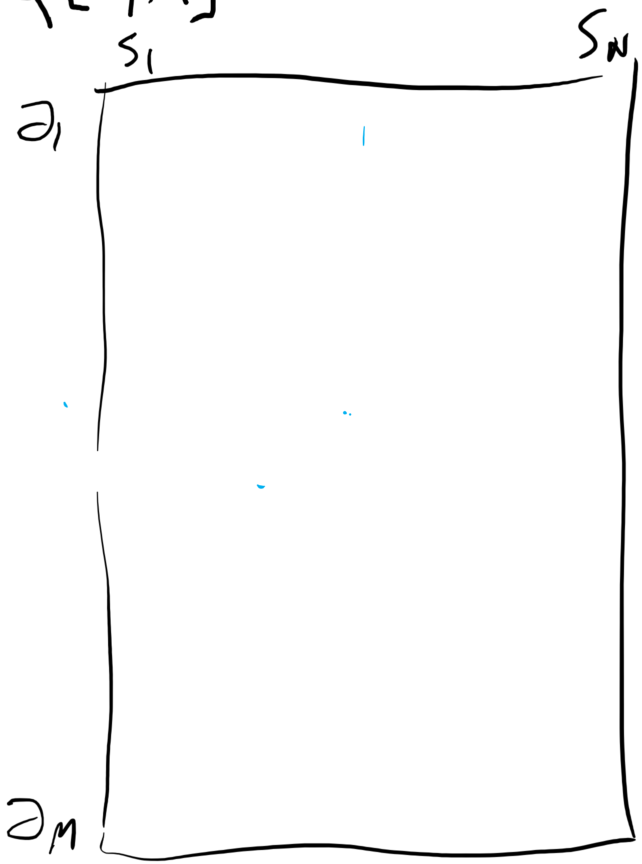
Feb, 1, 2021

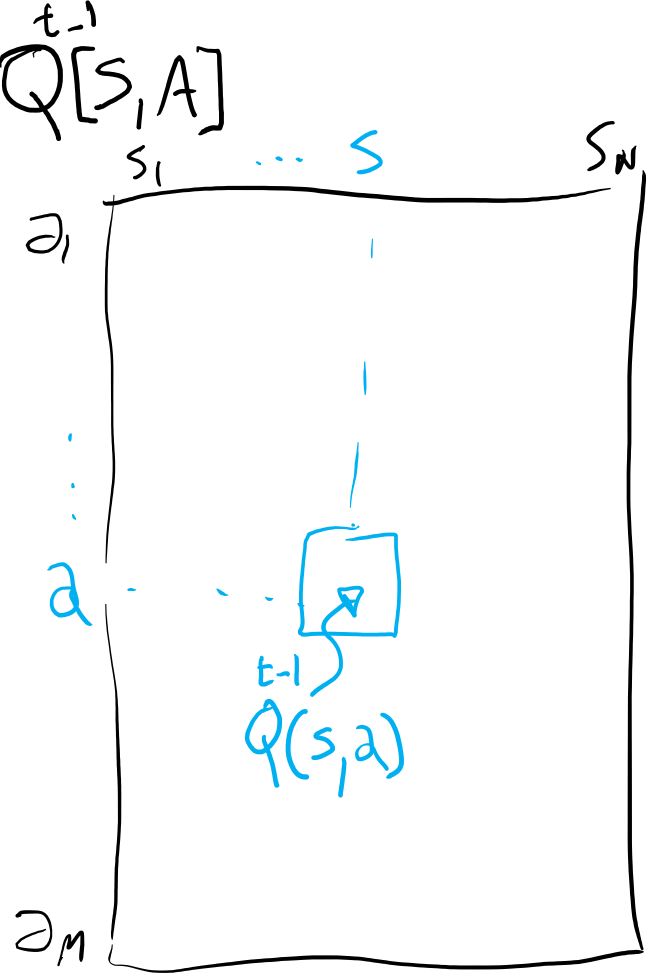
Lecture Overview

Finish Reinforcement learning

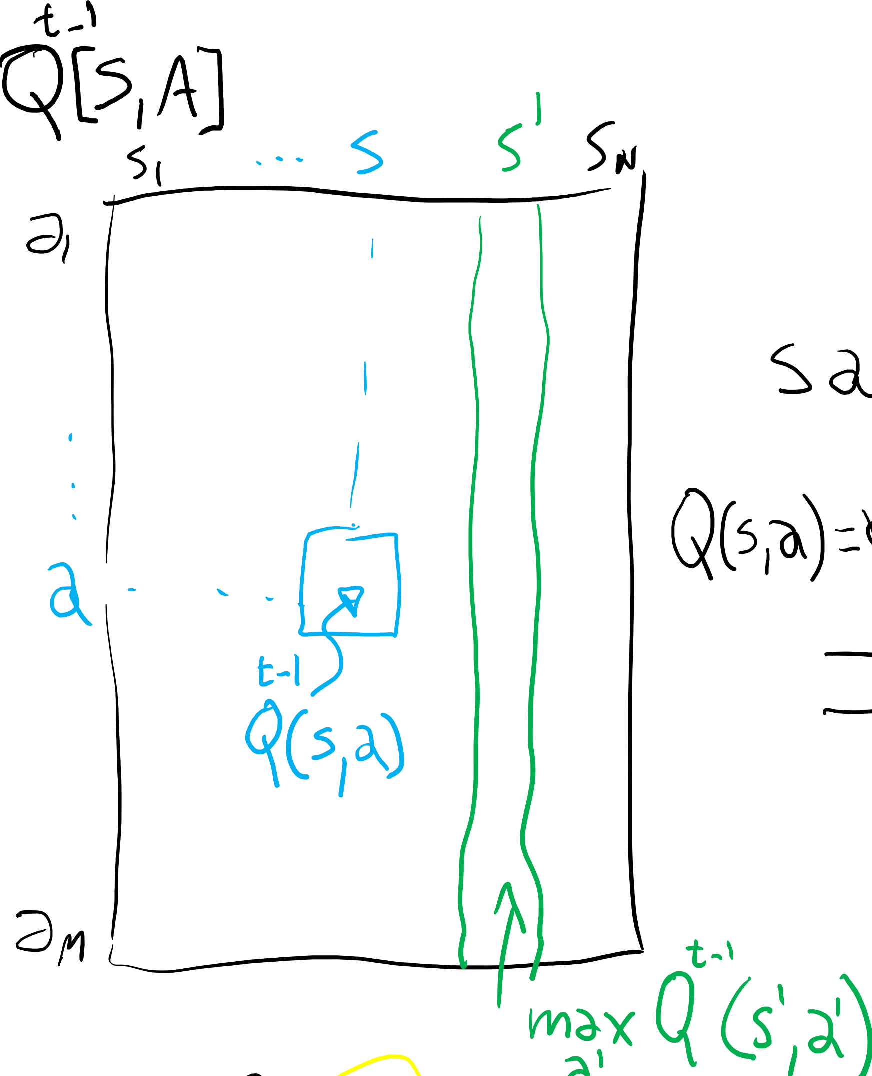
- **Exploration vs. Exploitation**
- On-policy Learning (SARSA)
- Scalability

$Q^{t-1}[S, A]$



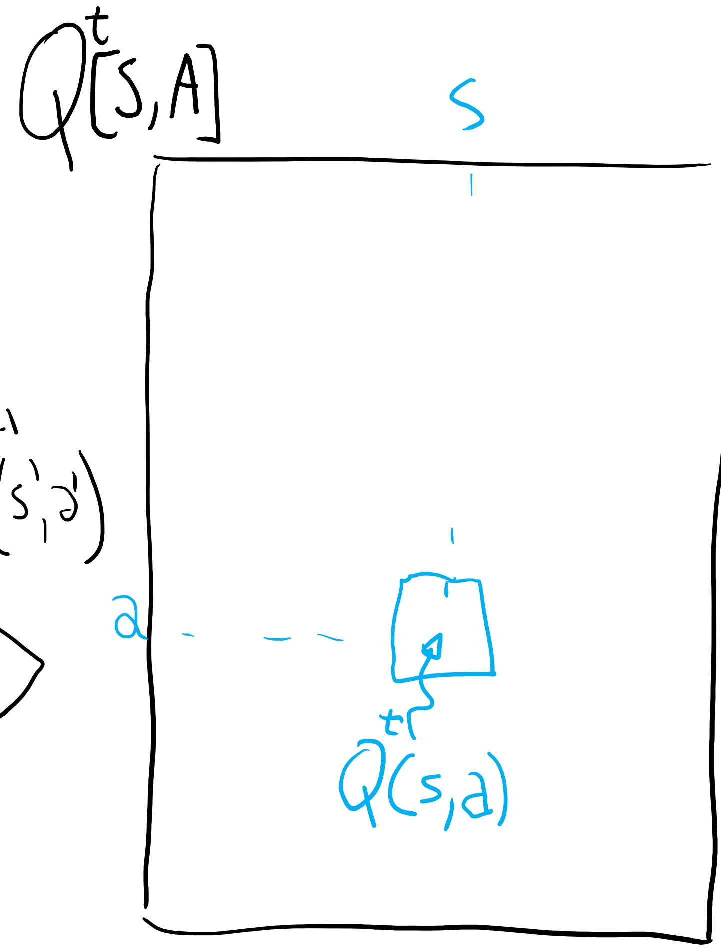


sars'



sars'

$$Q(s,a) = r + \gamma \max_{a'} Q^{t-1}(s',a')$$

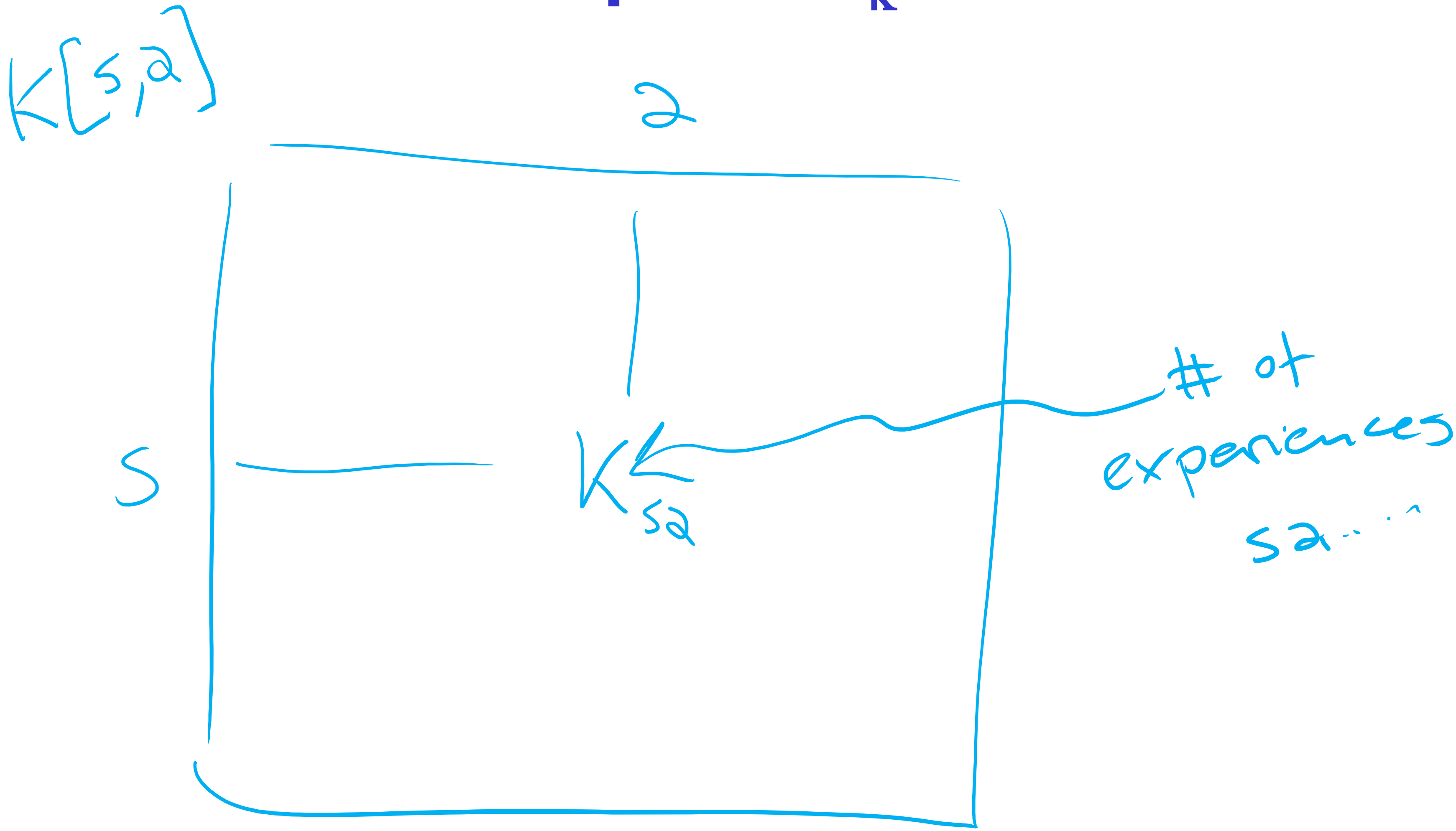


TD

$$A^t = A^{t-1} + \alpha_k \left(r^t - A^{t-1} \right)$$

$$Q^t(s,a) = Q^{t-1}(s,a) + \alpha_k \left(\left(r + \gamma \max_{a'} Q^{t-1}(s',a') \right) - Q^{t-1}(s,a) \right)$$

Also keep the α_κ



What Does Q-Learning learn

- ➢ Q-learning does not explicitly tell the agent what to do….

- ➢ Given the Q-function the agent can……

…. either exploit it or explore more….

Any effective strategy should be *greedy in the limit of infinite exploration* (**GLIE**)

- **Try each action an unbounded number of times**
- **Choose the predicted best action in the limit**

- We will look at two exploration strategies
  - ε-greedy
  - soft-max

ε-greedy

- Choose a **random action with probability ε** and choose **best action with probability 1- ε**

- First GLIE condition (try every action an unbounded number of times) is satisfied via the ε random selection

- What about second condition?
  - Select predicted best action in the limit.

- reduce ε overtime!

Soft-Max

- Takes into account improvement in estimates of expected reward function Q[s,a]
  - Choose action ***a*** in state ***s*** with a probability proportional to current estimate of **Q[s,a]**

$$\frac{e^{Q[s,a]}}{\sum_{a} e^{Q[s,a]}}$$

Soft-Max

- When in state **s**, Takes into account improvement in estimates of expected reward function Q[s,a] for all the actions
  - Choose action *a* in state *s* with a probability proportional to current estimate of Q[s,a]

$$\frac{e^{Q[s,a]}}{\sum_a e^{Q[s,a]}} \qquad \frac{e^{Q[s,a]/\tau}}{\sum_a e^{Q[s,a]/\tau}}$$

- τ (tau) in the formula above influences how randomly values should be chosen
  - if τ is high, >> Q[s,a]?

A. It will mainly exploit

B. It will mainly explore

C. It will do both with equal probability

Lecture Overview

Finish Reinforcement learning

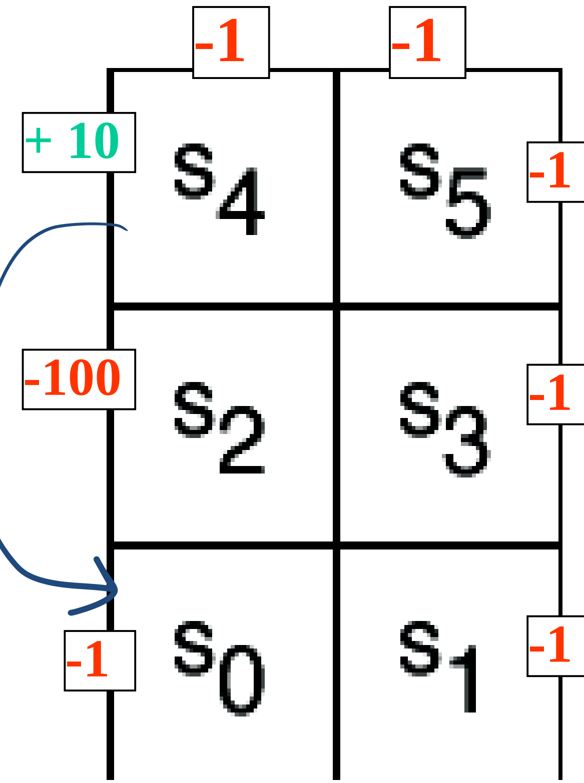
- Exploration vs. Exploitation
- **On-policy Learning (SARSA)**
- **RL scalability**

Learning before vs. during deployment

- Our learning agent can:
  A. act in the environment to learn how it works (before deployment)
  B. Learn as you go (after deployment)
- If there is time to learn before deployment, the agent should try to do its best to learn as much as possible about the environment
  - even engage in locally suboptimal behaviors, because this will guarantee reaching an optimal policy in the long run
- If learning while “at work”, suboptimal behaviors could be costly

Example

- Six possible states $<s_0,..,s_5>$
- 4 actions:
  - *UpCareful:* moves one tile up unless there is wall, in which case stays in same tile. Always generates a penalty of -1
  - *Left:* moves one tile left unless there is wall, in which case
    - ✓ stays in same tile if in s_0 or s_2
    - ✓ Is sent to s_0 if in s_4
  - *Right:* moves one tile right unless there is wall, in which case stays in same tile
  - *Up:* 0.8 goes up unless there is a wall, 0.1 like *Left*, 0.1 like *Right*

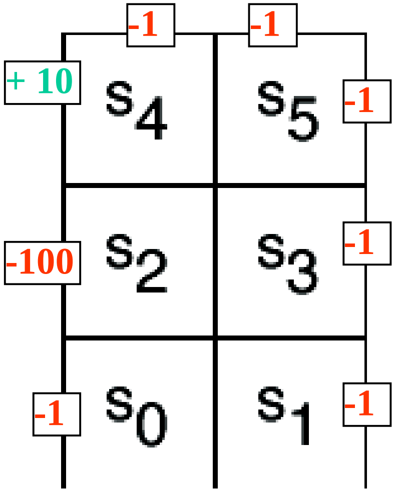


Reward Model:

- -1 for doing *UpCareful*
- Negative reward when hitting a wall, as marked on the picture
- *+10* for *left* in s_4

Example

- Consider, for instance, our sample grid game:
  - the optimal policy is to go *up* in S_0
  - But if the agent includes some exploration in its policy (e.g. selects 20% of its actions randomly), exploring in S_2 could be dangerous because it may cause hitting the -100 wall
  - No big deal if the agent is not deployed yet, but not ideal otherwise



- Q-learning would not detect this problem
  - It does *off-policy learning*, i.e., it focuses on the optimal policy
- *On-policy* learning addresses this problem

On-policy learning: SARSA

- On-policy learning learns the value of the policy being followed.
  - e.g., act greedily 80% of the time and act randomly 20% of the time
  - Better to be aware of the consequences of exploration has it happens, and avoid outcomes that are too costly while acting, rather than looking for the true optimal policy
- SARSA
  - So called because it uses <*state, action, reward, state, action*> experiences rather than the <*state, action, reward, state*> used by Q-learning
  - Instead of looking for the best action at every step**, it evaluates the actions suggested by the current policy**
  - Uses this info to revise it

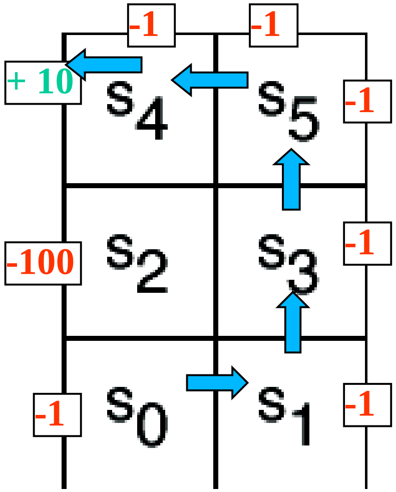
On-policy learning: SARSA

In Q-learning we assume that the agent in s' will follow the optimal policy….

$$Q[s,a] \leftarrow Q[s,a] + \alpha((r + \gamma \max_{a'} Q[s',a']) - Q[s,a])$$

- Given an experience <*s,a,r,s',a'* >, *SARSA* updates Q[s,a] seeing that the current policy has selected a'… so how we update?

$\langle s_0, right, 0, s_1, upCareful, -1, s_3, upCareful, -1, s_5, left, 0, s_4, left, 10, s_0, right\rangle$



$$Q[s,a] \leftarrow Q[s,a] + \alpha(r + \gamma Q[s',a'] - Q[s,a])$$

k=1

Q[s,a]	s_0	s_1	s_2	s_3	s_4	s_5
upCareful	0	0	0	0	0	0
Left	0	0	0	0	0	0
Right	0	0	0	0	0	0
Up	0	0	0	0	0	0

$$Q[s_0, right] \leftarrow Q[s_0, right] + \alpha_k(r + 0.9Q[s_1, UpCareful] - Q[s_0, right]);$$
$$Q[s_0, right] \leftarrow$$

$$Q[s_1, upCarfull] \leftarrow Q[s_1, upCarfull] + \alpha_k(r + 0.9Q[s_3, UpCareful] - Q[s_1, upCarfull]);$$
$$Q[s_1, upCarfull] \leftarrow$$

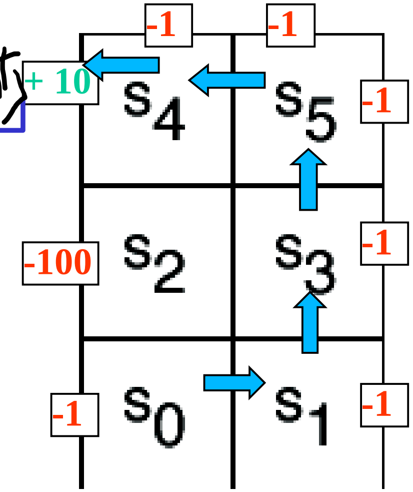
$$Q[s_3, upCarfull] \leftarrow Q[s_3, upCarfull] + \alpha_k(r + 0.9Q[s_5, Left] - Q[s_3, upCarfull]);$$
$$Q[s_3, upCarfull] \leftarrow 0 + 1(-1 + 0.9*0 - 0) = -1$$

$$Q[s_5, Left] \leftarrow Q[s_5, Left] + \alpha_k(r + 0.9Q[s_4, left] - Q[s_5, Left]);$$
$$Q[s_5, Left] \leftarrow 0 + 1(0 + 0.9*0 - 0) = 0$$

Only immediate rewards are included in the update, as with Q-learning

$$Q[s_4, Left] \leftarrow Q[s_4, Left] + \alpha_k(r + 0.9Q[s_0, Right] - Q[s_4, Left]);$$
$$Q[s_4, Left] \leftarrow 0 + 1(10 + 0.9*0 - 0) = 10$$

$\langle s_0, right, 0, s_1, upCareful, -1, s_3, upCareful, -1, s_5, left, 0, s_4, left, 10, s_0, right\rangle$



$$Q[s,a] \leftarrow Q[s,a] + \alpha(r + \gamma Q[s',a'] - Q[s,a])$$

k=2

Q[s,a]	s_0	s_1	s_2	s_3	s_4	s_5
upCareful	**0**	**-1**	0	**-1**	0	0
Left	0	0	0	0	**10**	**0**
Right	0	0	0	0	0	0
Up	0	0	0	0	0	0

$Q[s_0, right] \leftarrow Q[s_0, right] + \alpha_k(r + 0.9Q[s_1, UpCareful] - Q[s_0, right]);$

$Q[s_0, right] \leftarrow$

SARSA backs up the expected reward of the next action, rather than the max expected reward

$Q[s_1, upCarfull] \leftarrow Q[s_1, upCarfull] + \alpha_k(r + 0.9Q[s_3, UpCareful] - Q[s_1, upCarfull]);$

$Q[s_1, upCarfull] \leftarrow$

$Q[s_3, upCarfull] \leftarrow Q[s_3, upCarfull] + \alpha_k(r + 0.9Q[s_5, Left] - Q[s_3, upCarfull]);$

$Q[s_3, upCarfull] \leftarrow -1 + 1/2(-1 + 0.9*0 + 1) = -1$

$Q[s_5, Left] \leftarrow Q[s_5, Left] + \alpha_k(r + 0.9Q[s_4, left] - Q[s_5, Left]);$

$Q[s_5, Left] \leftarrow 0 + 1/2(0 + 0.9*10 - 0) = 4.5$

$Q[s_4, Left] \leftarrow Q[s_4, Left] + \alpha_k(r + 0.9Q[s_0, Right] - Q[s_4, Left]);$

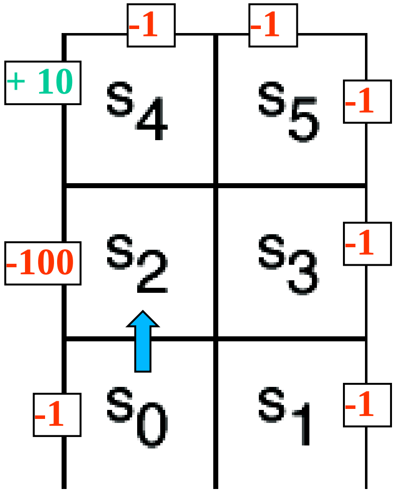
$Q[s_4, Left] \leftarrow 10 + 1/2(10 + 0.9*0 - 10) = 10$

Comparing SARSA and Q-learning

- For the little 6-states world

- Policy learned by **Q-learning** 80% greedy is to go *up* in s_0 to reach s_4 quickly and get the big +10 reward

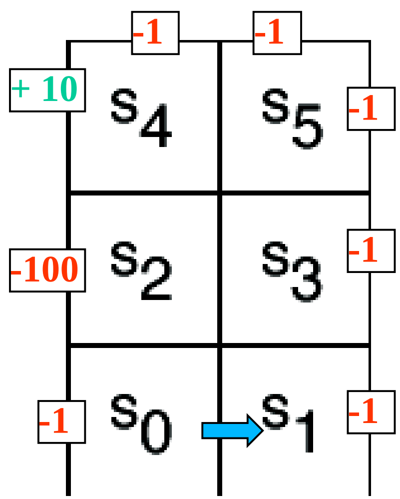
Iterations	Q[s_0,Up]	Q[s_1,Up]	Q[s_2,UpC]	Q[s_3,Up]	Q[s_4,Left]	Q[s_5,Left]
40000000	19.1	17.5	22.7	20.4	26.8	23.7



Comparing SARSA and Q-learning

- Policy learned by **SARSA** 80% greedy is to go *right* in s_0
- Safer because avoid the chance of getting the -100 reward in s_2
- but non-optimal => lower Q-values

Iterations	$Q[s_0,Right]$	$Q[s_1,Up]$	$Q[s_2,UpC]$	$Q[s_3,Up]$	$Q[s_4,Left]$	$Q[s_5,Left]$
40000000	6.8	8.1	12.3	10.4	15.6	13.2



SARSA Algorithm

This could be, for instance any ε-greedy strategy:
-Choose random ε times, and max the rest

begin

initialize $Q[S, A]$ arbitrarily

observe current state s

select action a using a policy based on Q

repeat forever:

carry out an action a

observe reward r and state s'

select action a' using a policy based on Q

$Q[s, a] \leftarrow Q[s, a] + \alpha\left(r + \gamma Q[s', a'] - Q[s, a]\right)$

$s \leftarrow s'$;

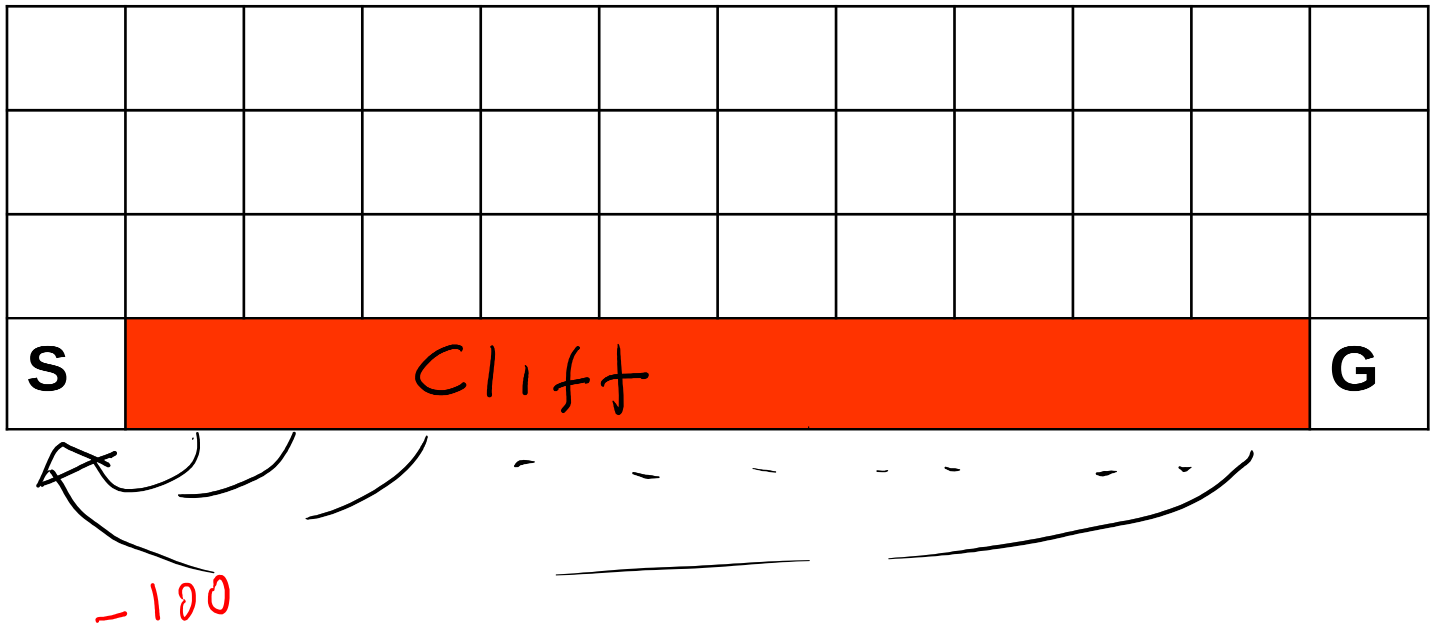
$a \leftarrow a'$;

end-repeat

end

Another Example

- Gridworld with:
  - Deterministic actions *up*, *down*, *left, right*
  - Start from **S** and arrive at **G** (terminal state with reward > 0)
  - **Reward is -1 for all transitions**, except those into the region marked “Cliff”
    - ✓ Falling into the cliff causes the agent to be sent back to start: **r = -100**



- With an **ε-greedy strategy (e.g., ε =0.1)**

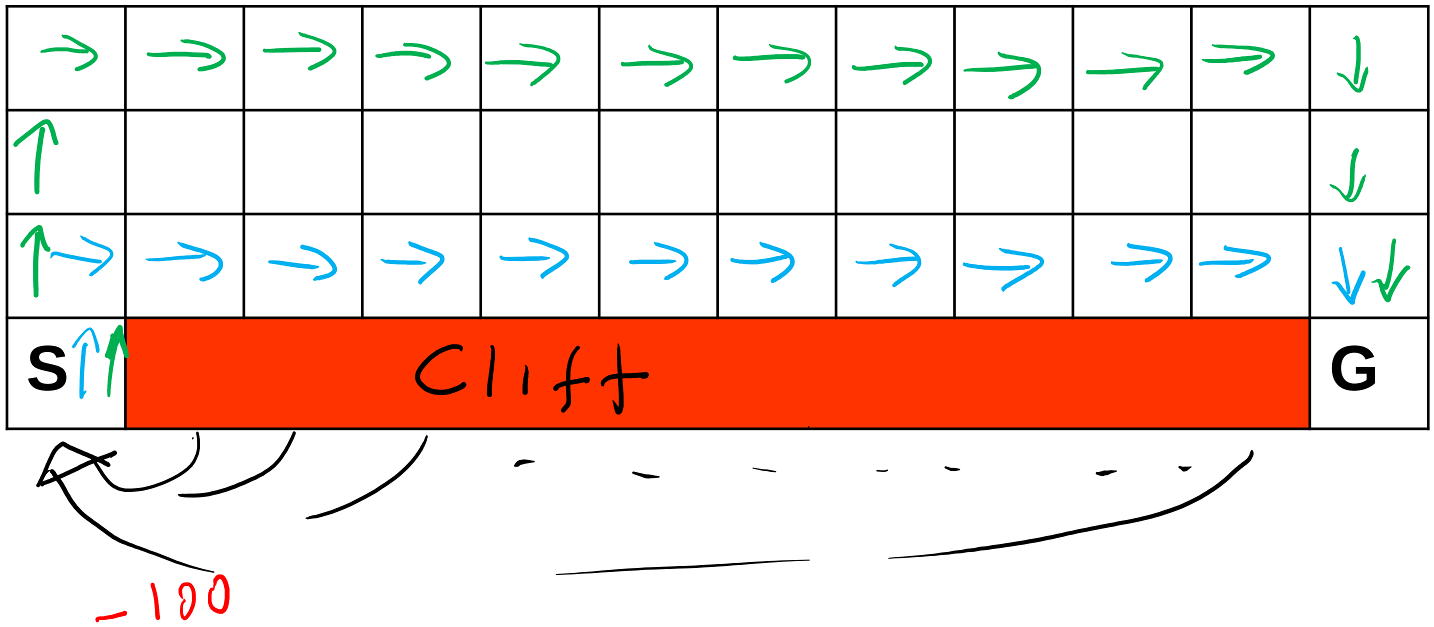
iclicker

A. SARSA will learn policy p1 while Q-learning will learn p2

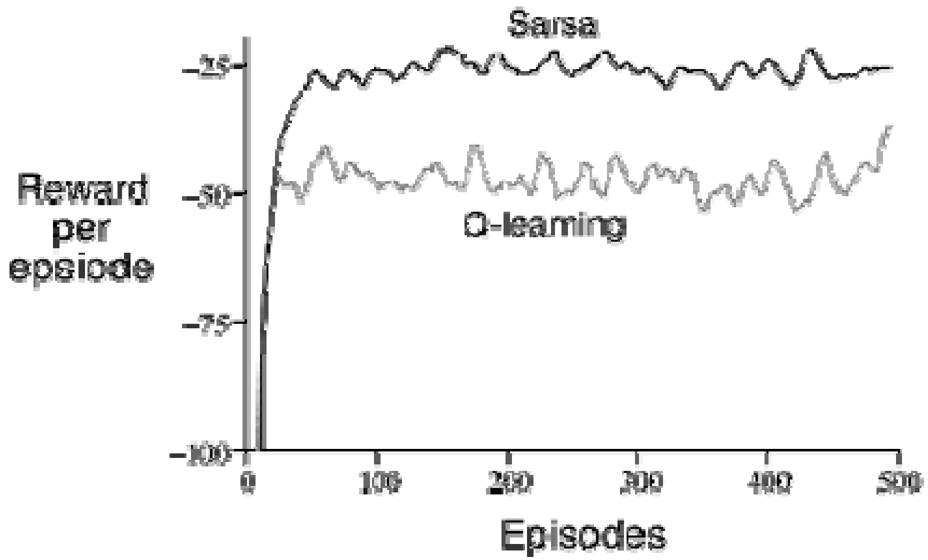
B. Q-learning will learn policy p1 while SARSA will learn p2

C. They will both learn p1

D. They will both learn p2



Q-learning vs. SARSA



- Q-learning learns the optimal policy, but because it does so without taking exploration into account, it does not do so well while the agent is exploring
  - It occasionally falls into the cliff, so its reward per episode is not that great
- SARSA has better on-line performance (reward per episode), because it learns to stay away from the cliff while exploring
  - But note that if $\varepsilon \rightarrow 0$, SARSA and Q-learning would asymptotically converge to the optimal policy

Final Recommendation

- If agent is **not deployed** it should do ….

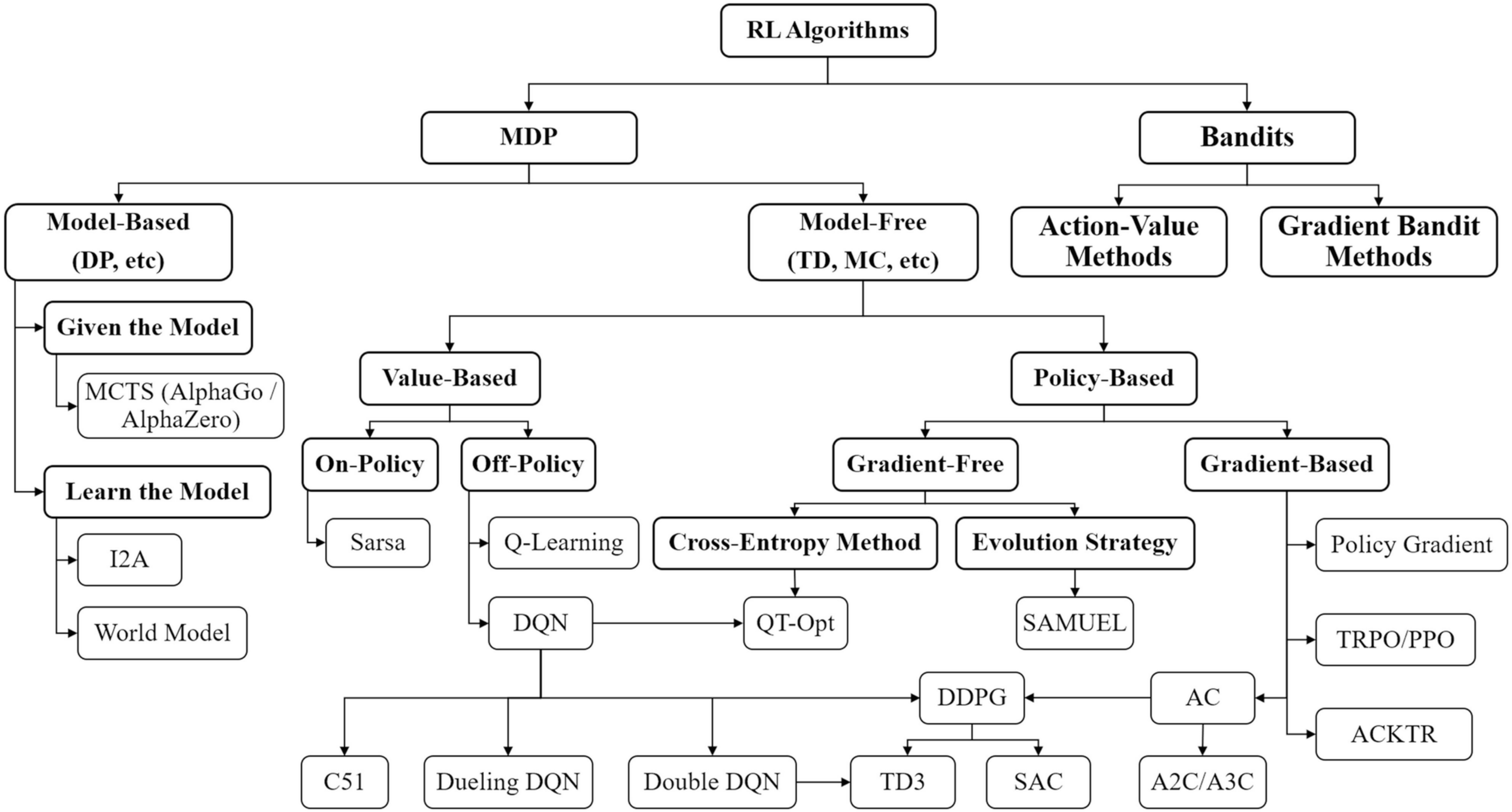
  random all the time (ε=1) and **Q-learning**

  - When Q values have converged then deploy

- If the agent is **deployed** it should

  - apply one of the explore/exploit strategies (e.g., ε=.5) and do **Sarsa**
  - Decreasing ε over time

NOT REQUIRED for 422! Map of reinforcement learning algorithms. Boxes with thick lines denote different categories, others denote specific algorithms



422 big picture

StarAI (statistical relational AI)

Hybrid: Det +Sto

Prob CFG **Parsing**

Prob Relational Models

Markov Logics

	Deterministic	**Stochastic**
Query	***Logics*** ***First Order Logics*** ***Ontologies*** • **Full Resolution** • **SAT**	***Belief Nets*** **Approx. : Gibbs** ***Markov Chains and HMMs*** **Forward, Viterbi….** **Approx. : Particle Filtering** ***Undirected Graphical Models*** ***Markov Networks*** ***Conditional Random Fields***
Planning		***Markov Decision Processes and Partially Observable MDP*** • **Value Iteration** • **Approx. Inference** ***Reinforcement Learning***

Applications of AI

Representation

Reasoning Technique

Learning Goals for today's class

➢You can:

- Describe and compare techniques to combine exploration with exploitation
- On-policy Learning (SARSA)
- Discuss trade-offs in RL scalability (not required)

TODO for Wed

- Read textbook 6.4.2
- Next research paper will be next Mon
- Practice Ex 11.B

- Assignment 1 due on Wed